Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Office

An Agency of Industry Canada

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed with Application for Patent Serial No: **2,335,521**, on February 9, 2001, by **STERGIOS V. ANASTASIADIS**, for "Distributed Media Server Architecture".

ERTIFIED COPY OF
IORITY DOCUMEN

[illegible]

Agent certificateur/Certifying Officer

February 18, 2002

Date

Canada

(CIPO 68)
01-12-00

OPIC CIPO

Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Office

An Agency of Industry Canada

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,335,521**, on February 9, 2001, by **STERGIOS V. ANASTASIADIS**, for "Distributed Media Server Architecture".

S. Grégoire

Agent certificateur/Certifying Officer

February 18, 2002

Date

Canada

OPIC CIPO

DISTRIBUTED MEDIA SERVER ARCHITECTURE

Field of Invention

This invention relates to media servers and in particular to a distributed media server architecture.

Background of the Invention

Explosive growth in online services has increased the demand for modular and efficient network server systems. System design complications coupled with excessive expectations from technological progress previously discouraged the development of media servers efficiently supporting video streams with variable bit rates.

Variable Bit Rate (VBR) video streams have been estimated to be smaller (by 40% or more) than Constant Bit Rate (CBR) streams of comparable quality. Correspondingly, media servers supporting VBR streams can be expected to concurrently serve more users than their CBR counterparts, due to reduced requirements for disk space, disk bandwidth, buffer space, and network bandwidth.

Nevertheless, most of the existing experimental or commercial media servers can only support CBR streams. Alternatively, they store VBR streams using either peak rate resource reservations that may reduce resource utilization but not increase server capacity, or statistical Quality of Service (QoS) guarantees that allow the system to occasionally be overloaded and discard data. The approach of retrieving VBR streams using constant rates, popular with lower bit rate streaming applications, may not solve the general problem either due to arbitrarily large playback initiation latency or client buffer space that can be required with higher quality streams.

The feasibility and potential advantages of deterministically supporting VBR streams over multiple disks have not been demonstrated yet. Online access to video streams of acceptable quality remains challenging due to their vast resource requirements. Predicted advances in optical and wireless telecommunication technology are only expected to increase the need for network servers with the capacity to support large numbers of users. According to recent estimates though, disk bandwidth increases at an order of magnitude slower than network link bandwidth. It has been estimated that disk bandwidth improves by a factor of ten every decade, while network link bandwidth improves by a factor of four every three years.

Existing video server systems, including those commercially available, only support streams with constant bit rate or peak rate resource reservations. On the other hand, variable bit rate (VBR) video encoding can achieve equally acceptable quality as constant bit rate encoding while having only 60% as high average bit rates. Furthermore, detailed per round resource reservation can increase the system throughput by more than a factor of two when compared to peak rate resource reservation for variable bit rate streams.

It is therefore an aspect of an object of the present invention for providing a distributed video server architecture, called Exedra, to reduce the requirements for disk space, disk bandwidth, buffer space, and network bandwidth.

Brief Description of the Drawings

In the accompanying drawings:

Figure 1 illustrates an Exedra media server in accordance with an embodiment of the present invention;

Figure 2 illustrates a stride-based allocation of disk space server in accordance with an embodiment of the present invention;

Figure 3 illustrates system modules in an implementation in accordance with an embodiment of the present invention;

Figure 4 illustrates a circular vector of dispatch queues in an implementation in accordance with an embodiment of the present invention;

Figure 5 illustrates buffer allocation performance in an implementation in accordance with an embodiment of the present invention;

Figure 6 illustrates disk space allocation in an implementation in accordance with an embodiment of the present invention;

Figure 7 illustrates disk time reservation efficiency in an implementation in accordance with an embodiment of the present invention; and

Figure 8 illustrates system scaling in an implementation in accordance with an embodiment of the present invention.

Detailed Description of the Preferred Embodiments

Referring to the drawings and initially to Figure 1, there is illustrated an Exedra media server in accordance with an embodiment of the present invention comprises transfer nodes, admission control nodes, local-area network, storage interconnect, and storage for schedule descriptors and stream data. Exedra is a distributed media server system based on standard off-the-self components for data storage and transfer. Video streams are stored on multiple disks, compressed according to the MPEG-2 specification, or any other encoding scheme that supports constant quality quantization parameters and variable bit rates. Multiple clients with appropriate stream decoding capability send playback requests to the server and receive stream data via a high-speed network, as shown in Figure 1.

The system operates using the server-push model. When a playback session starts, the server periodically sends data to the client until either the end of the stream is reached, or the client explicitly requests suspension of the playback. The server-push model facilitates quality of service enforcement at the server side, when compared to a client-pull model. The data transfers occur in rounds of fixed duration round. In each round, an appropriate amount of data is retrieved from the disks into a set of server buffers reserved for each active client. Concurrently, data are sent from the server buffers to the client through the network interfaces. Round-based operation is typically used in media servers in order to keep the reservation of the resources and the scheduling-related bookkeeping of the data transfers manageable.

The large amount of network bandwidth needed for this kind of service requires that the server consist of multiple components, connected to the high-speed network through different network interfaces. The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream and the resource management policy of the network. A policy is to send to the client during each round the amount of data that is needed for the decoding process of the next round. Alternately, any other policy that does not violate the timing requirements and buffering constraints of the decoding client is also acceptable.

In the Exedra media server, stream data are retrieved from the disks and sent to the clients through the Transfer Nodes. Both the admission control and the data transfers make use of stream scheduling information maintained as a set of schedule descriptors.

The stream data are stored across multiple disks. As shown in Figure 1, each disk is connected to a particular Transfer Node through the Storage Interconnect, which is one of either i) standard I/O channel (e.g. Small Computer System Interface), ii) network storage equipment (e.g. Fiber-Channel), or iii) a general purpose network (as with Network-Attached Secure Disks). Alternately, server functionality may be off loaded to network-attached disks.

The Transfer Nodes are computers responsible for scheduling and initiating all data transfers from the attached disks to the clients. Data arriving from the disks are temporarily stored in the Server Buffer memory of the Transfer Node before being sent to the clients. The system bus bandwidth (such as the Peripheral Component Interconnect) is a critical resource within each Transfer Node that essentially restricts the number and the capacity of the attached network and I/O channel interfaces.

Playback requests arriving from the clients are initially directed to an Admission Control Node, where it is decided whether sufficient resources exist to activate the requested playback session either immediately or within a few rounds. The computational complexity of the general stream scheduling problem is combinatorial in the number of streams considered for activation and the number of reserved resources. However, the practical assumption is that the users only expect to wait a limited number of rounds before actual playback starts. This limits the number of future rounds considered for playback initiation, and permits the use of a simpler scheduling algorithm with complexity linear in the number of rounds of each stream and the number of reserved resources.

The examples described in this document are solely for purposes of illustration and are not intended to limit the scope of the invention. Changes in form and substitution of equivalents are contemplated as circumstances may suggest or render expedient. Although specific terms have been employed herein, such terms are intended in a descriptive sense and not for purposes of limitation.

For example, with rounds of one second, a two hour stream, and a single node system with two disks and one network interface, then the maximum number of numerical comparisons required is equal to the stream length (7200) multiplied by the number of resources (4). This computation requirement corresponds to a worst case general case. However, this is significantly relaxed in practice, since not all the resources are involved in each round. If the test fails, it has to be repeated for each extra future round considered for playback initiation. Depending on the expected load and the required detail of resource reservation, the admission control process might still become a bottleneck. In that case, the admission control is distributed across multiple processors as shown in Figure 1, taking into account non-trivial concurrency control issues that arise. If a new playback request is accepted, commands are sent to the Transfer Nodes to begin the appropriate data accesses and transfers.

The amount of stream data that needs to be retrieved during each round from each disk is stored in a Schedule Descriptor. The descriptor also specifies the buffer space required and the amount of data sent to the client by the Transfer Nodes during each round. It is possible that two or more schedule descriptors are available for the same stream with distinct requirements. The scheduling information is generated when a stream is first stored and is used for both admission control and for controlling data transfers during playback. Since this information changes infrequently, alternately, it can be replicated to avoid potential bottlenecks.

In the present invention, there is provided a new form of disk space allocation, called stride-based allocation, in which disk space is allocated in large, fixed-sized chunks (strides) that are sequentially allocated on the disk surfaces. Strides are chosen larger than the maximum stream request size per disk during a round. This size is known in advance for stored streams.

Stride-based allocation has a number of advantages over schemes that are used in other systems. It sets an upper-bound on the estimated disk access overhead, since at most two partial stride accesses is required to serve the request of a stream on each disk in a round. It eliminates external fragmentation, while keeping internal fragmentation negligible, because of the large size of the streams, and because a stride may contain data of more than one round as shown in Figure 2. When a stream is retrieved, only the requested amount of data is fetched to memory and not the entire stride.

As illustrated in Figure 2, a stride-based allocation of disk space is shown on one disk. A stream is stored in a sequence of generally non-consecutive fixed-size strides with a stride possibly containing data of more than one round. Sequential requests of one round are smaller than the stride size and thus require at most two partial stride accesses.

A mathematical abstraction of the resource requirements is necessary for scheduling purposes. Consider a system consisting of N network interfaces, D disks, and Q transfer nodes.

The stream Network Striping Sequence, Sn, of length Ln defines the amount of data, Sn(i,u), 1=< i=< Ln, 0=< u =< N-1, that the server sends to a particular client through network interface u during round i. Similarly, the Buffer Striping Sequence Sb of length Lb = Ln + 1 defines the server buffer space required on node q, Sb(i,q), 0=< i=< Lb, 0=< q =< Q-1 during round i. Each stride is a sequence of logical blocks with fixed size Bl, which is multiple of the physical sector size Bp of the disk. Both disk transfer requests and memory buffer reservations are specified in multiples of the block size Bl. The Disk Striping Sequence Sd of length Ld = Ln determines the amount of data, Sd(i,k), that are retrieved from disk k, 0=< k =< D-1, in round i, 0=< i=< Ld - 1.

Assuming disk k, 0=< k =< D-1, has edge to edge seek time TfullSeek^k, single track seek time TtrackSeek^k, average rotational latency TavgRot^k, and minimum internal transfer rate Rdisk^k. The stride-based disk space allocation policy enforces an upper bound of at most two disk arm movements per disk for each client per round. The total seek distance is also limited using a CSCAN disk scheduling policy. Assuming that seek latency is always a linear function of the seek distance. For short seeks, seek latency is known to depend on the square root of the seek distance though. Head settling time is accounted for through the single-track seek time parameter of the disk specification. The accuracy of these approximations are quantified later.

Let Mi be the number of active streams during round i of the system operation. Also the playback of stream j, 1=< j=< Mi, is initiated at round lj of system operation. Then, the total access time on disk k in round i of the system operation has an upper-bound of:

Tdisk(i,k) = 2TfullSeek^k + 2Mi*(TtrackSeek^k + TavgRot^k)\\ & + $\Sigma^{M}_{j=1}$ Sd^j (i – lj, k)/Rdisk^k

Sd^j is the disk striping sequence of client j. T^kfullSeek is counted twice due to the disk arm movement from the CSCAN policy, while the factor two in the second term is due to the stride-based allocation. The reservations of transfer time on each network interface and buffer space on each transfer node are more straightforward, and are based on the Network Striping Sequence and Buffer Striping Sequence, respectively.

With Variable-Grain Striping, stream files are stored on disks such that the data retrieved during a round for a client are always accessed from a single disk round-robin. Comparison with alternative striping techniques has shown significant performance benefits when using Variable-Grain Striping, and this is the method that is used in the present invention.

Example

The example is described for the purposes of illustration and is not intended to limit the scope of the invention.

A single-node multiple-disk media server prototype was designed and built in order to evaluate the resource requirements of alternative stream scheduling techniques. The modules are implemented in about 12,000 lines of C++/ Pthreads code on AIX4.2. The code is linked to the University of Michigan DiskSim disk simulation package, which incorporates advanced features of modern disks, such as on-disk cache and zones, for obtaining disk access time measurements. The code is also able to directly use hardware disks through their raw interface for full data transfers. The stream indexing metadata are stored in the Unix file system as regular files, and during operation are kept in main memory.

The basic modules of the media server include file naming, resource reservation, admission control, logical to physical metadata mapping, buffer management, and disk and network transfer scheduling are illustrated in Figure 3. With appropriate configuration parameters, the system is able to operate in several modes to allow different levels of detail for evaluation analysis. In Admission Control mode, the system receives playback requests, does admission control and resource reservation, but no actual data transfers take place. In Simulated Disk mode, all the modules become functional, and disk request processing takes place using the specified DiskSim disk array. In Full Operation mode, the system accesses hardware disks and transfers data to clients.

The admission control module uses circular vectors of sufficient length to represent the allocated disk time, network time, and buffer space, respectively. On system startup, all elements of disk time vectors are initialized to 2*TfullSeek, while the network time and buffer space vector elements are set to zero. When a new stream request arrives, the admission control is performed by checking the requirements of the stream against currently available resources. In particular, the total service time of each disk in any round may not exceed the round duration, the total network service time on any network interface may not exceed the round duration, and the total occupied buffer space on any transfer node may be no larger than the corresponding server buffer capacity.

If the admission control test is passed, then the resource sequences of the stream are added to the corresponding system vectors managed by the module, and the stream is scheduled for playback. In addition, notification records for the accepted request are inserted into the corresponding dispatch queues (generally residing on each transfer node) at the appropriate offset from the current round. When an upcoming round becomes current, the notification records are used for activating the stream and starting its data transfers as illustrated in Figure 4 where a circular vector of dispatch queues keeps track of admitted streams yet to be activated. The dispatch queue consists of notification records for activating the streams in the corresponding rounds.

Stream metadata management is organized in a layer above disk scheduling. It is responsible for disk space allocation during stream recording, and for translating stream file offsets to physical block locations during playback. The stream metadata are maintained as regular files in the host OS (of each transfer node, in the general case), while stream data are stored separately on dedicated disks. The storage space of the data disks is organized in strides, with a bitmap that has a separate bit for each stride. A single-level directory is used for mapping the identifier of each recorded stream into a direct index of the corresponding allocated strides. A separate directory of this form exists for each disk.

When a stream is striped across multiple disks, a stream file is created on each data disk. Each transfer request received by the metadata manager specifies the starting offset in the corresponding stream file and the number of logical blocks to be accessed. With the help of the stream index, each such request is translated to a sequence of contiguous disk transfers, each specifying the starting physical block location and the number of blocks. From the stride-based disk space allocation, it follows that each logical request will be translated into at most two physical contiguous disk transfers.

The decision to create a separate metadata manager for each disk has an advantage of application to general disk array organizations, including those consisting of heterogeneous disks. Although the handling of heterogeneous devices may not be necessary in limited size traditional storage systems, it might prove crucial for the incremental growth and survivability of large scalable media storage installations.

In order to keep system performance predictable and unbiased from particular disk geometry features, some control was exercised on the disk space allocation pattern. In particular, disk zoning could possibly lead to excessively optimistic or pessimistic data access delays, if most allocation were in the outer or inner cylinders of the disks. Similarly, contiguous allocation could lead to lower than expected delays in some special cases (such as when streams are stored on a single disk with a very large on-disk cache). However, low-level disk geometry is generally not disclosed by the disk manufacturers, therefore, strides for a stream within each disk were allocated to distribute them across all the zones of the disk.

The disk management layer is responsible for passing data transfer requests to the disks, after the necessary translation from logical stream offsets to physical block locations in the above layers.

In the dual-queue CSCAN disk scheduling used herein, the operation of each disk is managed by a separate pair of priority queues, called Request Queue and

Service Queue, respectively. The two queues, although structurally equivalent, play different roles during each round. At the beginning of each round, data transfer requests for the current round are added asynchronously into the request queue of each disk, where they are kept sorted in increasing order of their starting sector location.

When all the requests have been gathered (and the corresponding disk transfers of the previous round completed), the request queue of each disk is swapped with the corresponding service queue. Subsequently, requests from the service queue are synchronously submitted to the raw disk interface for the corresponding data transfers to occur. The two-queue scheme prevents new requests from getting service before those of the previous round complete. This keeps the system operation more stable in the rare (yet possible) case that the disk busy time in a round slightly exceeds the round duration.

When swapping the two queues, the service queue becomes request queue and remains empty until the beginning of the next round. Although a single priority queue for each disk would seem sufficient, there is a rare (yet possible) case where the disk busy time in a round slightly exceeds the round duration. Then, with a naive design using a single queue, new incoming requests could postpone (potentially indefinitely) the service of requests from the previous round starting at the innermost edge of a disk. Instead, the two-queue scheme prevents new requests from getting service before those of the previous round complete, thus keeping the system operation more stable.

Sequence of steps during the disk service of I/O requests using dual-queue CSCAN. For the special case of simulated disks, no actual data transfers occur and the requests are used only for disk access time estimation. Requests corresponding to different disks are submitted, processed, and terminate concurrently at a simulated time granularity enforced by the simulator. Whenever a disk request completes, the next request waiting to be served in the current round is synchronously submitted, similarly to the hardware disk case.

The buffer management module keeps the server memory organized in fixed size blocks of Bl bytes each, where Bl is the logical block size introduced earlier. Buffer space is allocated in groups of consecutive blocks. From experiments with raw interface disk accesses, we found that non-contiguity of the memory buffers could penalize disk bandwidth significantly on some systems. Although this might be attributed to the way that scatter/gather features of the disk controller are used by these systems, we found the allocation contiguity easy to enforce.

For the allocation of buffer blocks, a bitmap structure was used with an interface that supports block group requests. Deallocations are allowed on a block by block basis, even though entire block groups are acquired during allocation. This last feature allows more aggressive deallocations.

In this design, previously accessed data was not cached, as is typically done in traditional file and database systems. Although related research has developed data caching algorithms for constant rate streams, similar support for variable bit rate streams introduce several complications for evaluation purposed, especially in the admission control process. Instead, it was assumed that data transfers are done independently for each different playback.

Paging of buffer space is prevented by locking the corresponding pages in main memory. Although several Unix versions (e.g. HP-UX, Irix, Solaris) and Linux make the mlock system call available for this purpose, AIX does not. Instead, the pinu kernel service is exported through a loadable kernel module.

The configuration input provides the basic parameters of each device that is used. This includes the disk and metadata file name, the maximum and track seek time, the average rotation latency, the minimum internal transfer rate. Other configuration parameters include the available network bandwidth along with the server and client buffer space, as required by the different policies.

The schedule management module is responsible for generating data transfer schedules, where each schedule specifies the amount of data accessed from the disks, stored in server buffer and transmitted over the network during each round. The schedule manager accepts as input actual MPEG-2 Unix files (or their frame size traces), along with the prefetching and striping scheme that can be used. The prefetching schemes can make use of the buffer space that is available at the server or client side, depending on the policy. However, no prefetching scheme is used in this prototype example. The striping schemes specify the disks and the amount of data that is to be accessed during each round.

Performance measurements were made on an IBM RS/6000 two-way SMP workstation with 233 MHz PowerPC processors running AIX4.2. The system was configured with 256 MB physical memory, and a fast wide SCSI controller to which a single 2GB disk was attached, containing both the system and paging partitions. The stream data are stored on two 4.5GB Seagate Cheetah ST-34501W disks (see Table 1) attached to a separate ultra wide SCSI controller. Note that one megabyte (megabit) is considered equal to 2^20 bytes (bits), except for the measurement of transmission rates and disk storage capacities where it is assumed to be equal to 10^6 bytes (bits) instead. Although storage capacity can reach 73GB in the latest models, the performance numbers of the above two disks are typical of today's high-end drives.

Table 1 Features of the SCSI disks used in the prototype example.

Seagate Cheetah ST-34501W	
Data Bytes per Drive	4.55 GB
Average Sectors per Track	170
Data Cylinders	6,526
Data Surfaces	8
Zones	7
Buffer Size	0.5 MB
Track to Track Seek(read/write)	0.98/1.24 msec
Maximum Seek(read/write)	18.2/19.2 msec
Average Rotational Latency	2.99 msec
Internal Transfer Rate Inner Zone to Outer Zone Burst Inner Zone to Outer Zone Sustained	 122 to 177 Mbit/s 11.3 to 16.8 MB/s
External Transfer Rate	40 MB/s

Six different variable bit rate MPEG-2 streams of 30 minutes duration each were used for this evaluation. Each stream had 54,000 frames with a resolution of

720x480 and 24 bit color depth, 30 frames per second frequency, and a IB^2PB^2 PB^2PB^2PB^2 15 frame group of pictures structure. The encoding hardware used allowed the generated bit rate to take values between 1Mbit/s and 9.6Mbit/s. Although the Main Profile Main Level MPEG-2 specification allows bit rates up to 15Mbit/sec, there is a typical point of diminishing returns (no visual difference between original and compressed video) at 9Mbit/sec. The DVD specification sets a maximum allowed MPEG-2 bitrate of 9.8Mbit/sec. Statistical characteristics of the clips are given in Table 2, where the coefficients of variation (of bytes per round) lie between 0.028 and 0.383, depending on the content type. The MPEG-2 decoder from the MPEG Software Simulation Group was used for stream frame size identification.

Table 2: Characteristics of six MPEG-2 video streams of 30 minutes duration each that were used. The coefficient of variation shown in the last column changes according to the content type.

Content Type	Avg Bytes per rnd	Max Bytes per rnd	CoV per rnd
Science Fiction	624935	1201221	0.383
Music Clip	624728	1201221	0.366
Action	624194	1201221	0.245
Talk Show	624729	1201221	0.234
Adventure	624658	1201221	0.201
Documentary	625062	625786	0.028

Unless otherwise stated, the logical block size Bl was set equal to 16 KB, while the physical sector size Bp was 512 bytes. The stride size Bs in the disk space allocation was set to 2 MB. The total memory buffer size was set to 64 MB, organized in fixed size blocks of 16 KB. In this evaluation, data retrieved from the disks are discarded (copied from the buffer to the null device at the appropriate round), leaving protocol processing and contention for the network outside the scope of the present evaluation. By not including the network protocol overhead in the measurements that follow, it allowed the demonstration of the exact cost of the disk transfers involved, which is the main focus here. On the other hand, evaluation using the loopback interface gave results within 5% of those reported. The round time was set equal to one second.

Assuming that playback initiation requests arrive independently of one another, according to a Poisson process. The system load can be controlled by setting the mean arrival rate λ of playback initiation requests. The maximum possible service rate μ, expressed in streams per round for streams of data size Stot bytes, is equal to μ = D*Rdisk*Tround/Stot. Correspondingly, the system load ρ, is equal to ρ = λ/μ =< 1, where λ =< λmax = μ. The definition of ρ is used in this evaluation.

When a playback request arrives, the admission control module checks whether available resources exist for every round during playback. The test considers the exact data transfers of the requested playback for every round and also the corresponding available disk transfer time, network transfer time and buffer space in the system. If the request cannot be initiated in the next round, the test is repeated for each round up to [1/λ] rounds into the future, until the first round is found, where the requested playback can be started with guaranteed sufficiency of resources. Checking [1/λ] rounds into the future achieves most of the potential system capacity. If not accepted, the request is discarded rather than being kept in a queue.

Sample executions are repeated until the half-length of the 95% confidence interval on the performance measure of interest lies within 5% of the estimated mean value. The basic performance objective was to maximize the average number of active playback sessions that can be concurrently supported by the server.

This evaluation examined the potential effects of the buffer organization on the disk throughput; the implications of the disk space allocation parameters to the disk bandwidth utilization, and compare resource reservation statistics to actual utilization measurements; and finally, demonstrating that the system throughput scales linearly with the amount of resources made available, under the disk striping policy that was used.

In respect of Figure 5, in a) when a separate disk transfer is used for each buffer block, disk throughput depends critically on the block size; in b) grouping multiple block transfers into a single call by using Readv() cuts by more than 50% the achieved disk throughput; and in c) invoking a single Read() for each request keeps disk throughput consistently high, independently of the buffer block size.

The performance of the buffer allocation policy was evaluated using a synthetic benchmark that developed for this purpose. The disk throughput was measured for different sizes of I/O requests and degrees of contiguity in the buffer space allocated for each request. Disk requests of a specific size are initiated at different locations uniformly distributed across the disk space. Data are transferred through the raw disk interface to pinned memory organized in blocks of fixed size Bl.

As illustrated in Figure 5(a), there is depicted the average throughput, when a separate read() call is invoked for each buffer block corresponding to a request. Both the block size and the size of the request are varied. When the block size is increased from 4 KB to 64 KB, disk throughput changes by a factor of three across the different request sizes. When the request size varies from 64 KB to 4 MB for a particular block size, the throughput increases by more than a factor of two.

As illustrated in Figure 5(b), there is depicted the previous Figure 5(a) measurements by using the readv() system call instead. It takes as parameters the pointer to an array of address-length buffer descriptors along with the size of the array. The array size is typically limited to a small number of buffer descriptors (e.g. IOV_MAX =16 in AIX and Solaris). For each I/O request, the required number of readv() calls is used, with the array entries initialized to the address and length of each buffer block. Although, the expected improved performance due to the increased amount of information supplied with each readv() call to the OS, the measured throughput was less than half of what was measured with read(). Proper explanation for this would probably require internal knowledge of the AIX device drivers. An additional limit is also imposed to the I/O performance due to the small value of the IOV_MAX.

As illustrated in Figure 5(c), there is depicted the previous Figure 5(a) measurements by using only a single read() call for each request, similar to the way I/O requests are served in this prototype. The logical block size still determines the granularity of the disk transfer sizes and the buffer space (de)allocation. This policy requires contiguously located buffer blocks in the virtual address space. As expected the sensitivity to the block size Bl disappears. Note that the achieved performance is only slightly higher than that of figure Bs(a) with large blocks. However, the block size itself cannot be arbitrarily large; otherwise the benefit from multiplexing requests of different sizes drops, which eventually reduces the number of accepted streams. Since the average size of the disk transfers is about 625,000 bytes in our MPEG-2 clips, from these experiments, it is expected that the disks are able to operate at average throughput higher than 11 MB/s, which is consistent with the achievable sustained rate of 11.3 MB/s advertised in the disk specification.

For this system that contiguity in the buffer space allows a relatively small block size to be chosen that guarantees both the support of a large number of streams and efficient disk transfers. This simplifies the performance tuning of the system. One disadvantage is the complexity introduced by having to manage buffer block ranges instead of fixed buffers. In addition, buffer space fragmentation requires a number of buffers to remain unused (no more than 10-15% of the total buffer space, in this evaluation).

Arguably, disk access efficiency would improve if the disk space corresponding to each request were allocated contiguously, requiring a single disk head movement instead of a maximum of two as incurred by stride-based allocation. This was investigated by measuring disk bandwidth utilization when retrieving streams allocated on a disk using different stride sizes, while still keeping the stride size larger than the stream requests in a round (as per the original constraint). The achieved stream throughput is based on the resource reservations as noted previously, and remains the same across different stride sizes. As was explained before, the stream strides are approximately uniformly distributed across the disk space in order to prevent disk geometry biases.

Figure 6 shows the measured bandwidth utilization of a single disk configuration when retrieving different streams. The system load was set equal to ρ=80%, and the statistics were gathered over a period of 2,000 rounds after an initial warm up of 500 rounds. One important observation from these plots is that disk utilization drops as the stride size is increased from 2 MB to 16 MB. This is not surprising, since a larger stride size reduces disk head movements, and improves disk efficiency overall. Increasing the stride size from 2 MB to 16 MB reduces only marginally (2-3%) the disk bandwidth utilization across different stream types. Therefore, the expected benefit from either large strides, or contiguous disk space allocation, would be limited. A single-disk configuration was used with load ρ=80%.

However, Figure 6 shows that the total improvement in disk utilization does not exceed 2-3%. This percentage does not justify using larger strides (and increasing the unused storage space at the last stride of each stream). Instead, it indicates that stream disk accesses are dominated by useful data transfers rather than mechanical overhead. More generally, in an environment of multiple streams striped across several disks, the expected benefit from contiguous disk space allocation would be limited. Reduction in disk actuator overhead as a result of technology advances will only make this argument stronger.

For the following evaluations, the buffer block size was fixed to Bl=16 KB and the stride size to Bs=2 MB. In a system with 2 disks and 64 MB buffer memory, we compare the reserved and measured resource utilizations across different stream types. The resource reservations are based on the analytical estimations previously noted, and are intended to be accurate predictors of the corresponding measurements in the system. The system load was set to 80%, and gather statistics for a period of 2,000 rounds after a warm-up of 500 rounds. The higher loads only led to more rejected streams (not shown here), and do not significantly increase the system utilization. The average number of active streams in the above measurement period was roughly between 20 and 25 depending on the stream type.

Typically, the measured busy time in each round was less than or within a few milliseconds of the total disk time reserved. In only a small percentage of rounds (less than 1%) the discrepancy was higher, and this is hard to avoid completely, due to mechanical and other kinds of unexpected overhead. However, all the discrepancies are hidden from the client with an extra round added to the playback initiation latency. Other than that, stable prolonged system operation at high loads was achieved. The corresponding processor utilization hardly exceeded 5% on our SMP system. The processor utilization is expected to be higher when network protocol processing is included.

In respect of Figure 7, in a two-disk configuration with load ρ=80%, the measured disk utilization is balanced between the two disks. On each disk, the reserved disk utilization bounds relatively tightly (is only higher by about 5%) the measured disk utilization.

As illustrated in Figure 7, there is provided the fraction of the measurement period, during which each of the two disks was busy, and the corresponding fraction of reserved time. It was noted that the load is equally balanced across the two disks. (This observation remained valid when striping streams across larger disk arrays as well, which has important scalability implications.) In addition, the reserved busy fraction does not exceed by more than 5% the corresponding measured busy time. Hence, the admission control procedure offers quality of service guarantees, without disk bandwidth underutilization.

Each of the buffer blocks allocated for a data transfer is marked busy at the beginning of the round when the disk access occurs. It is not released until its last byte is sent over the network, in some subsequent round. On the other hand, resource reservation during admission control reserves the size of each buffer for the duration of the rounds that it spans. In the particular setup used here, no network transfers occur, essentially simulating an infinite bandwidth network. Fast network transfers imply early release of buffer blocks at the beginning of the round of the last-byte network transfer. Slower network transfers can delay block releases until the end of the round of the last-byte network transfer. In general, depending on the speed of the network subsystem and the network scheduling policy, it is expected that the measured buffer utilization to lie somewhere between the half and the total reserved buffer space fraction.

In order to investigate the scaling properties of this design, the system was used in Admission Control mode, where resource reservation occurs as with the previous experiments, but without any corresponding data transfers. This allows the study of system performance scalability, when additional system resources are available in the assumed hardware configuration.

As illustrated in Figure 8, there is shown the sustained number of active streams that can be supported with increasing number of disks across different stream types. The statistics were gathered during 6,000 rounds following a warmup period of 3,000 rounds. The figure shows that the number of streams increases in proportion to the number of disks used. This is a direct consequence of the load balancing property of Variable-Grain Striping. In addition, Figure 8 shows how performance also depends on the variability of data transfers across different rounds, which is different for each stream type (Table 2).

The number of accepted streams is projected to increase linearly as more disks are added to the system and the rest of the hardware resources increase proportionally. For these experiments, we run the system in Admission Control mode with the load at ρ=80%.

One of the better known media servers is the Tiger fault-tolerant video fileserver by Bolosky et al. It supports distributed storage of streams with constant bit rates only. The Fellini storage system by Martin et al. uses a client-pull model for accessing CBR/VBR stream data and does resource reservation based on the worst case requirements of each stream.

In the continuous media file server proposed by Neufeld et al., detailed resource reservation is done for each round, but the study focuses on storing data of an entire stream on a single disk. The Symphony multimedia file system by Shenoy et al. integrates data of different types on the same platform, with admission control based on peak rate assumptions. The present invention, instead, is customized for storage of stream data in order to maximize efficiency.

The RIO storage system by Muntz et al. is designed to handle several different data types, including video streams. The admission control is based on statistics, and the stream blocks are randomly distributed across different disks for load balancing. Instead, the present invention used deterministic admission control in order to achieve both balanced load and high bandwidth utilization across multiple disks, as demonstrated using actual MPEG-2 streams over SCSI disks.

An early design of distributed data striping is described by Cabrera and Long. Their data striping and resource reservation policies do not take into account special requirements of variable bit rate streams, however. In addition, striped data pass through an intermediate node before being sent to the clients. The present invention avoided such an approach for improved scalability.

In the design of FFS, McKusick et al. argue that chaining together kernel buffers would allow accessing contiguous blocks in a single disk transaction, and more than double the disk throughput. At that time, throughput was limited by processor speed however, and changes in the device drivers were also necessary for adding this feature.

Although stride-based allocation seems similar to extent-based and other allocation methods, one basic difference is that strides have fixed size. More importantly, when a stream is retrieved, only the requested amount of data is fetched to memory and not the entire stride, which is sequentially allocated on the disk surfaces.

The Exedra distributed media server architecture, and the described the details of a single-node multiple-disk prototype that as implemented have been described. The advantages of the present invention include the separation of metadata management for each disk to greatly simplify the structure of the system, and is capable of handling the case of heterogeneous disks; the dual-queue CSCAN disk scheduling method added stability to the system operation; contiguous allocation of memory buffers simplified performance tuning, while stride-based allocation kept the disk bandwidth utilization high without adding the complexity that contiguous disk space allocation would require. The resource reservation scheme matched relatively tightly the measured resource utilization. The sustained number of active streams increased linearly as more resources were added in the assumed configuration.

Although preferred embodiments of the invention have been described herein, it will be understood by those skilled in the art that variations may be made thereto without departing from the scope of the invention.

References

[1] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Disk Striping Scalability in the Exedra Media Server. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 2001). (to appear).

[2] Birrel, A. D., and Needham, R. M. A Universal File Server. *IEEE Transaction on Software Engineering* 6, 5 (Sept. 1980), 450–453.

[3] Bolosky, W. J., Barrera, J. S., Draves, R. P., Fitzgerald, R. P., Gibson, G. A., Jones, M. B., Levi, S. P., Myhrvold, N. P., and Rashid, R. F. The Tiger Video Fileserver. In *Intl. Work. on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan, Apr. 1996), pp. 97–104.

[4] Cabrera, L.-F., and Long, D. D. E. Swift: Using Distributed Disk Striping to Provide High I/O Data Rates. *Computing Systems* 4, 4 (1991), 405–436.

[5] Chang, E., and Zakhor, A. Cost Analyses for VBR Video Servers. *IEEE Multimedia* (Wint. 1996), 56–71.

[6] Clark, T. *Designing Storage Area Networks.* Addison-Wesley, Reading, Mass., 1999.

[7] Ganger, G. R., Worthington, B. L., and Patt, Y. N. The DiskSim Simulation Environment: Version 2.0 Reference Manual. Tech. Rep. CSE-TR-358-98, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, Dec. 1999.

[8] Garofalakis, M. N., Ioannidis, Y. E., and Ozden, B. Resource Scheduling for Composite Multimedia Objects. In *Very Large Data Bases Conf.* (New York, NY, Aug. 1998), pp. 74–85.

[9] Gibson, G. A., Nagle, D. F., Amiri, K., Butler, J., Chang, F. W., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., and Zelenka, J. A Cost-Effective, High-Bandwidth Storage Architecture. In *Conf. Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, Oct. 1998), pp. 92–103.

[10] Gringeri, S., Shuaib, K., Egorov, R., Lewis, A., Khasnabish, B., and Basch, B. Traffic Shaping, Bandwidth Allocation, and Quality Assessment for MPEG Video Distribution over Broadband Networks. IEEE Network, 6 (Nov/Dec 1998), 94–107.

[11] *The IBM Dictionary of Computing.* McGraw-Hill, New York, NY, 1994.

[12] Jones, M. B. The Microsoft Interactive TV System: An Experience Report. Tech. Rep. MSR-TR-97-18, Microsoft Research, 1997. ftp://ftp.research.microsoft.com/pub/tr/tr-97-18/tr-97-18.html.

[13] Lakshman, T. V., Ortega, A., and Reibman, A. R. VBR Video: Tradeoffs and Potentials. *Proceedings of the IEEE* 86, 5 (May 1998), 952–973.

[14] Martin, C., Narayanan, P. S., Ozden, B., Rastogi, R., and Silberschatz, A. The Fellini Multimedia Storage System. In *Multimedia Information Storage and Management* (Boston, MA, 1996), S.M.Chung, Ed., Kluwer Academic Publishers.

[15] McKusick, M. K., Joy, W. N., Leffler, S., and Fabry, R. S. A Fast File System for UNIX. *ACM Transactions on Computer Systems* 2, 3 (Aug. 1984), 181–197.

[16] McVoy, L., and Kleiman, S. R. Extent-like Performance from a Unix File System. In *USENIX Winter Technical Conference* (Dallas, TX, 1991), pp. 33–43.

[17] MPEG Software Simulation Group. *MPEG-2 Encoder/Decoder, Version 1.2*, 1996.

[18] Muntz, R., Santos, J. R., and Berson, S. A Parallel Disk Storage System for Real-Time Multimedia Applications. *International Journal of Intelligent Systems* **13**, 12 (Dec. 1998), 1137–1174.

[19] Neufeld, G., Makaroff, D., and Hutchinson, N. Design of a Variable Bit Rate Continuous Media File Server for an ATM Network. In *IS&T/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1996), pp. 370–380.

[20] RealNetworks, Inc. *Working with RealProducer 8 Codecs.* Seattle, WA, June 2000. Technical Blueprint.

[21] Reddy, A. L. N., and Wijayaratne, R. Techniques for improving the throughput of VBR streams. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1999), pp. 216–227.

[22] Ruemmler, C., and Wilkes, J. An Introduction to Disk Drive Modeling. *Computer* **27**, 3 (Mar. 1994), 17–28.

[23] Sen, S., Dey, J., Kurose, J., Stankovic, J., and Towsley, D. Streaming CBR transmission of VBR stored video. In *SPIE Symposium on Voice, Video and Data Communications* (Dallas, TX, Nov. 1997), pp. 26–36.

[24] Shenoy, P. J., Goyal, P., Rao, S. S., and Vin, H. M. Symphony: An Integrated Multimedia File System. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 1998), pp. 124–138.

[25] Shenoy, P. J., Goyal, P., and Vin, H. M. Issues in Multimedia Server Design. *ACM Computing Surveys* **27**, 4 (Dec. 1995), 636–639.

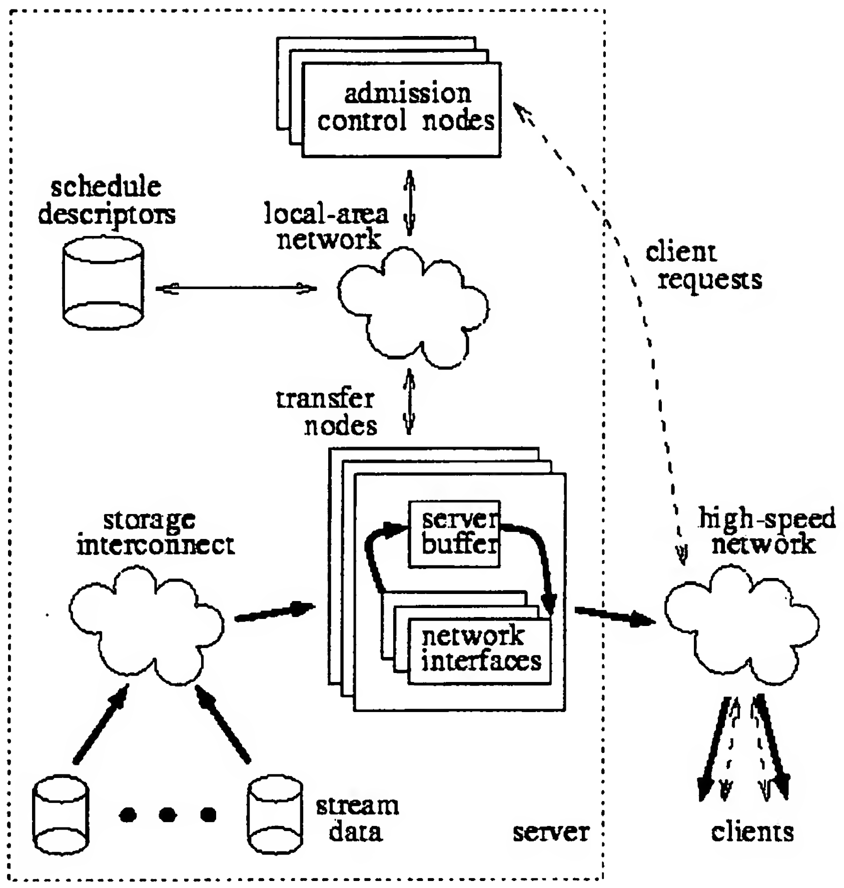

admission control nodes
schedule descriptors
local-area network
client requests
transfer nodes
storage interconnect
server buffer
network interfaces
high-speed network
stream data
server
clients

Figure 1

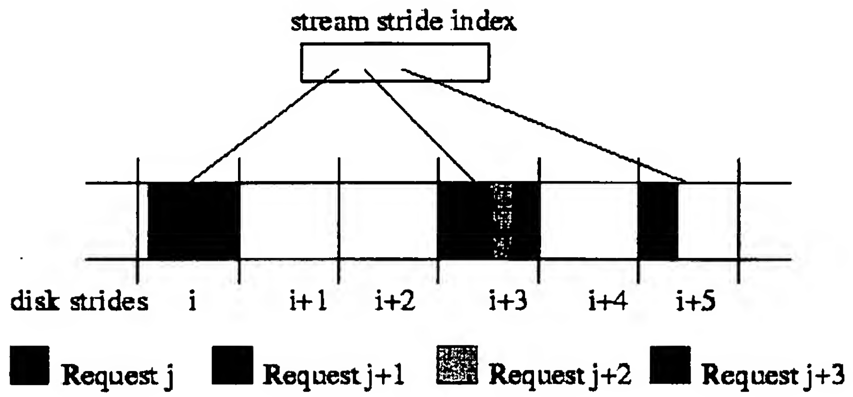

stream stride index
disk strides
i
i+1
i+2
i+3
i+4
i+5
Request j
Request j+1
Request j+2
Request j+3

Figure 2

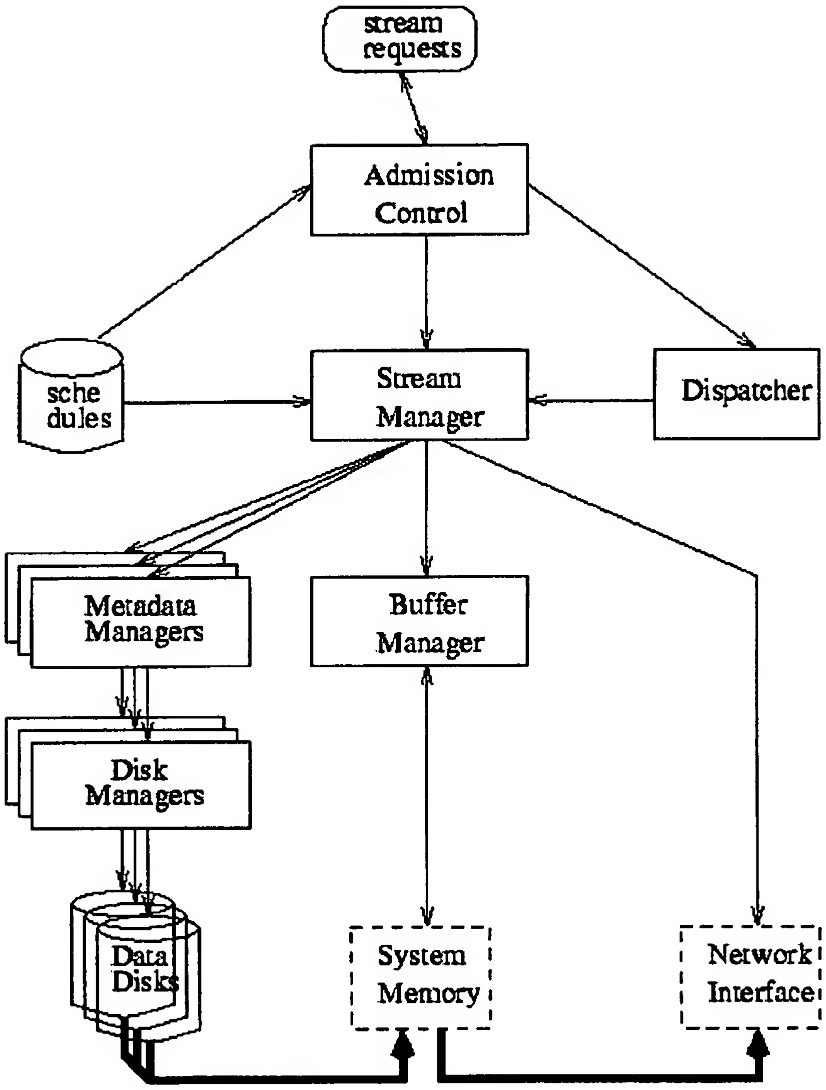

stream requests
Admission Control
sche dules
Stream Manager
Dispatcher
Metadata Managers
Buffer Manager
Disk Managers
Data Disks
System Memory
Network Interface

Figure 3

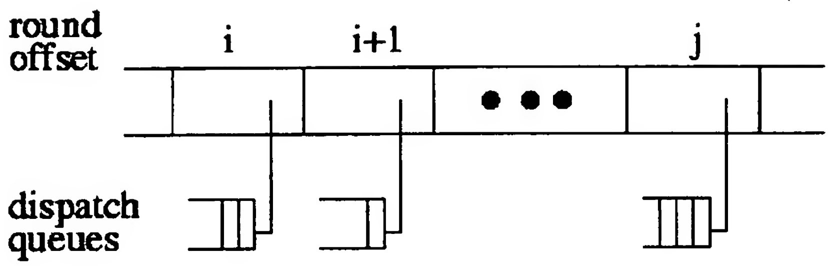


Figure 4

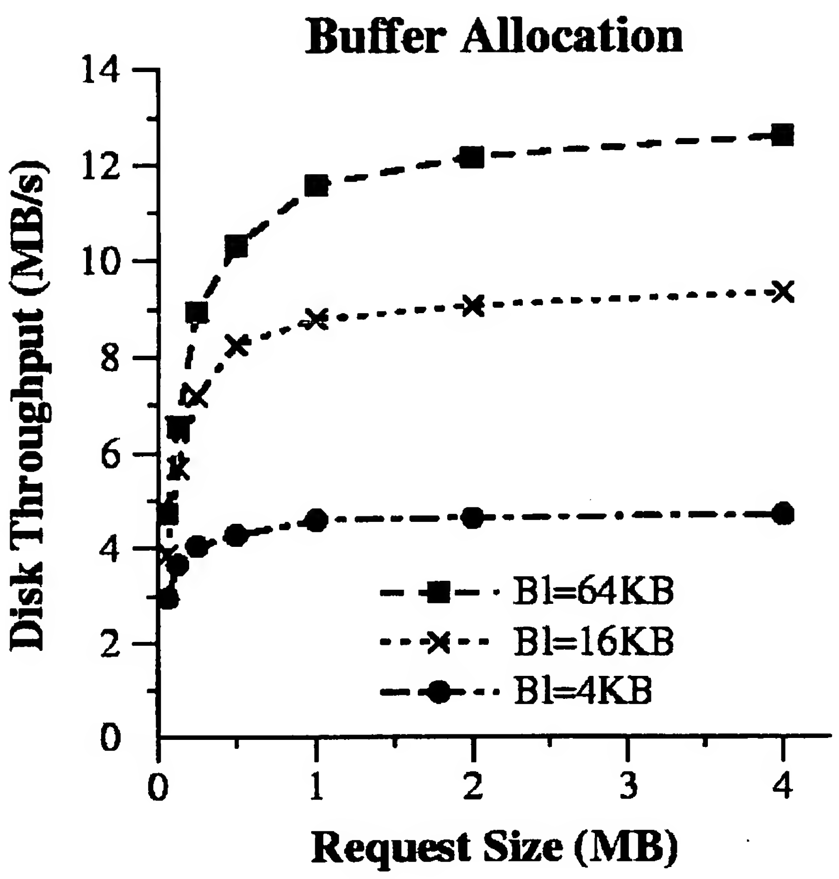
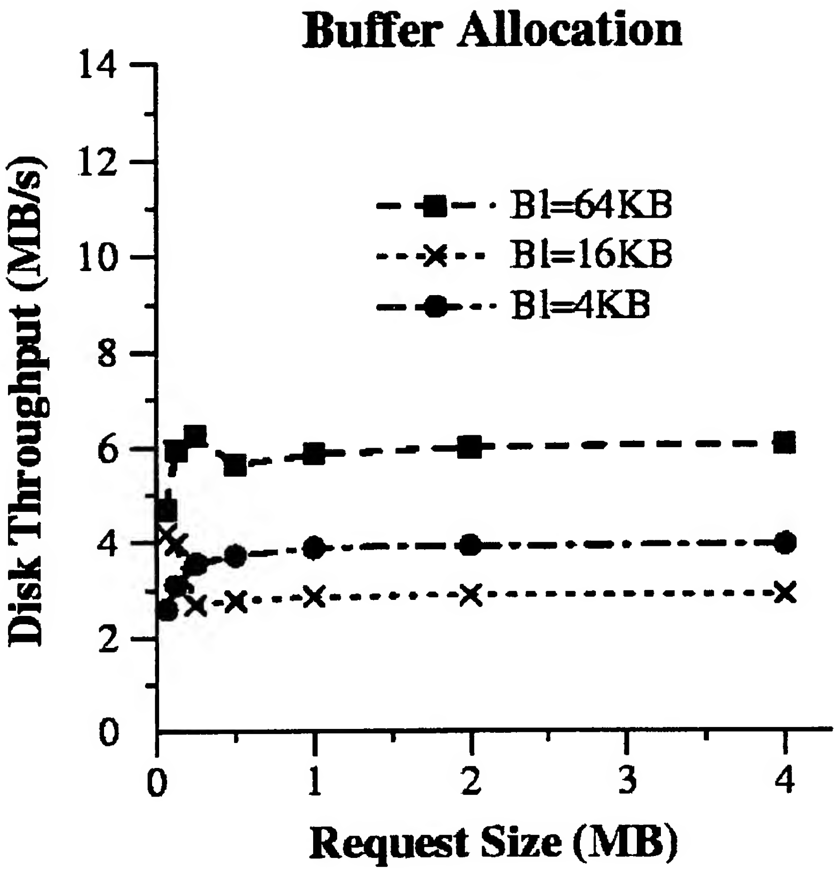

Buffer Allocation
Disk Throughput (MB/s)
Request Size (MB)
0
2
4
6
8
10
12
14
0
1
2
3
4
Bl=64KB
Bl=16KB
Bl=4KB

Figure 5(a)



Buffer Allocation
Disk Throughput (MB/s)
Request Size (MB)
Bl=64KB
Bl=16KB
Bl=4KB
0
2
4
6
8
10
12
14
0
1
2
3
4

Figure 5(b)

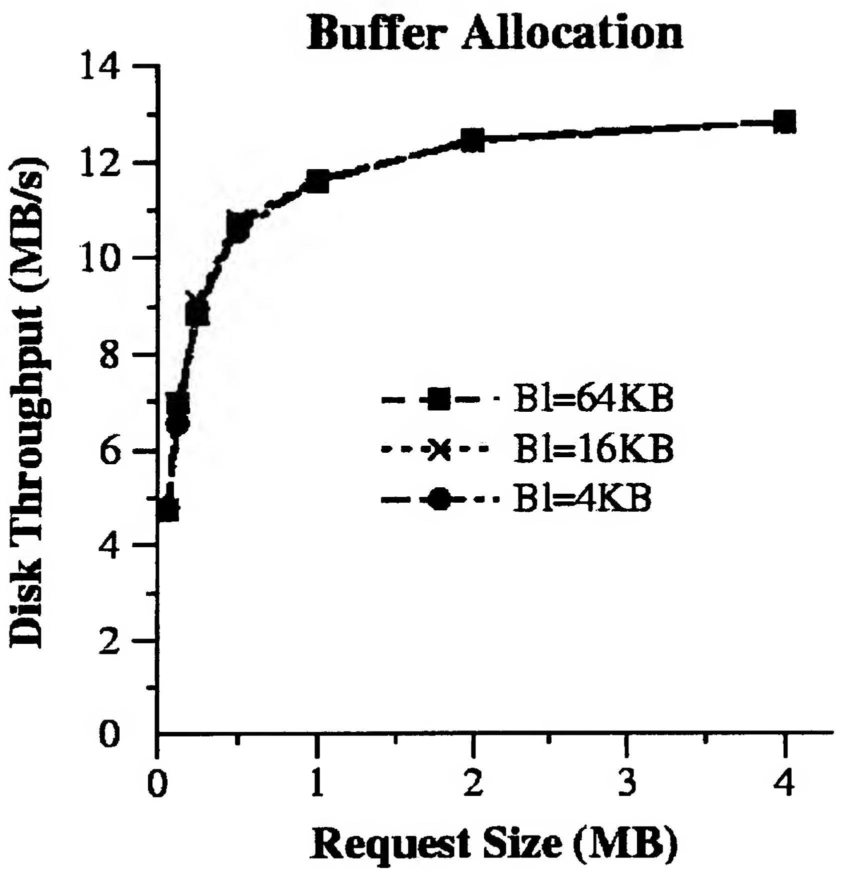

Buffer Allocation
Disk Throughput (MB/s)
Request Size (MB)
0
2
4
6
8
10
12
14
0
1
2
3
4
Bl=64KB
Bl=16KB
Bl=4KB

Figure 5(c)

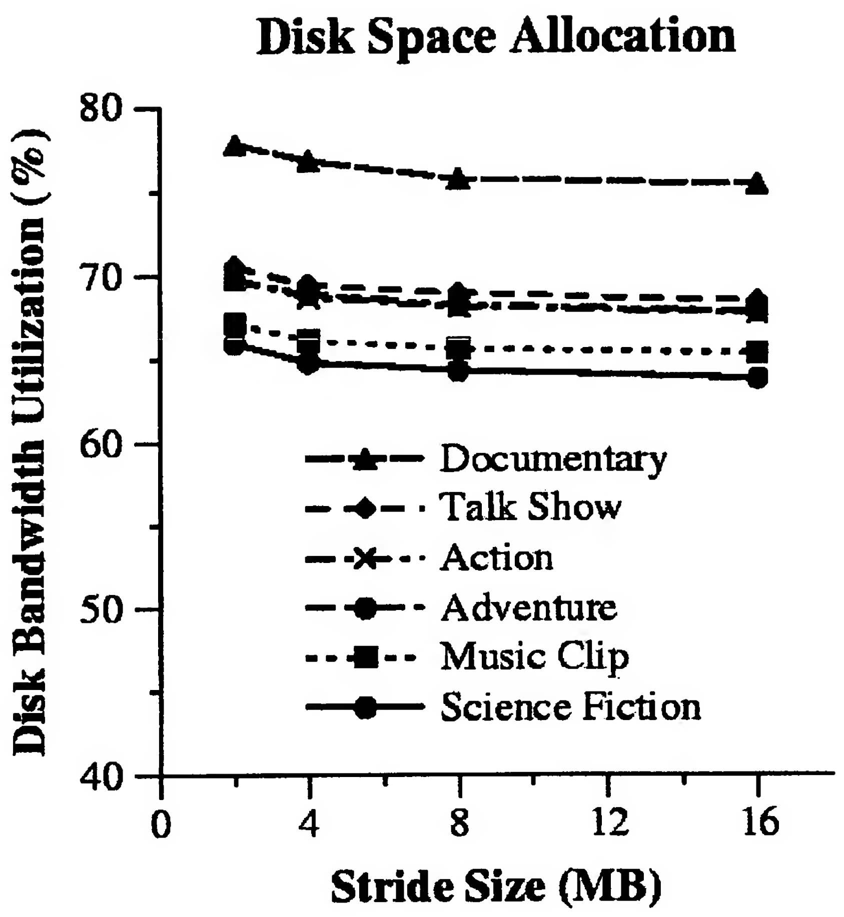

Disk Space Allocation
Disk Bandwidth Utilization (%)
80
70
60
50
40
0
4
8
12
16
Stride Size (MB)
Documentary
Talk Show
Action
Adventure
Music Clip
Science Fiction

Figure 6

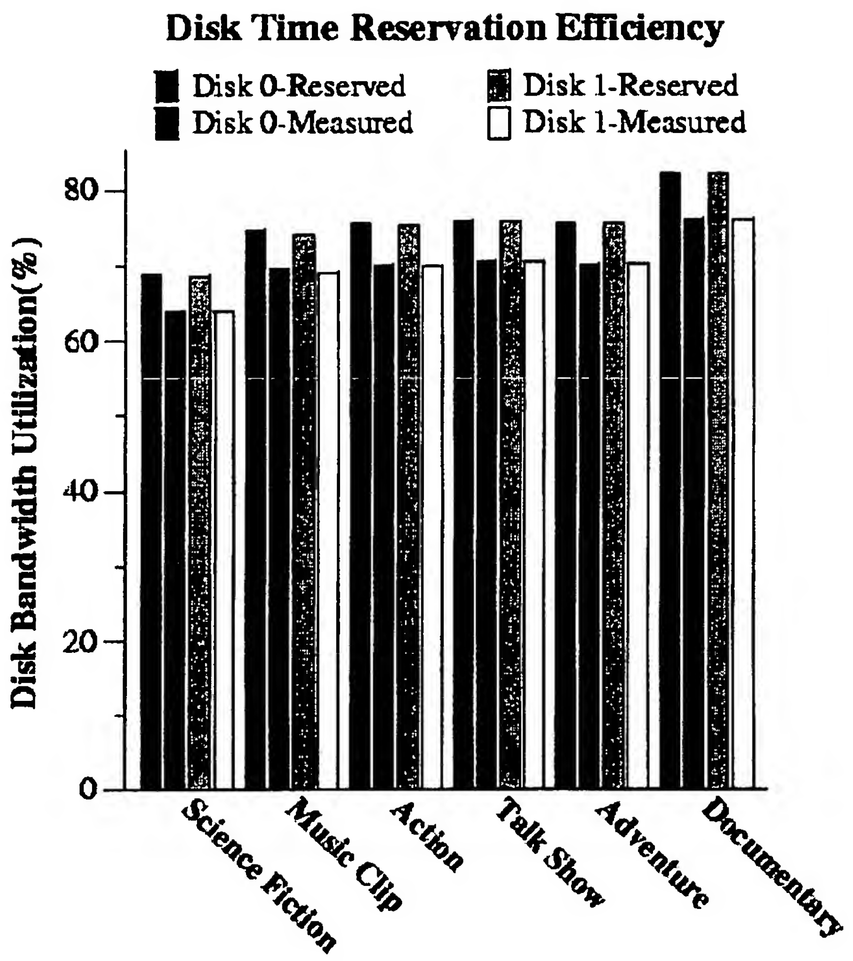

Disk Time Reservation Efficiency
Disk 0-Reserved
Disk 0-Measured
Disk 1-Reserved
Disk 1-Measured
Disk Bandwidth Utilization(%)
80
60
40
20
0
Science Fiction
Music Clip
Action
Talk Show
Adventure
Documentary

Figure 7

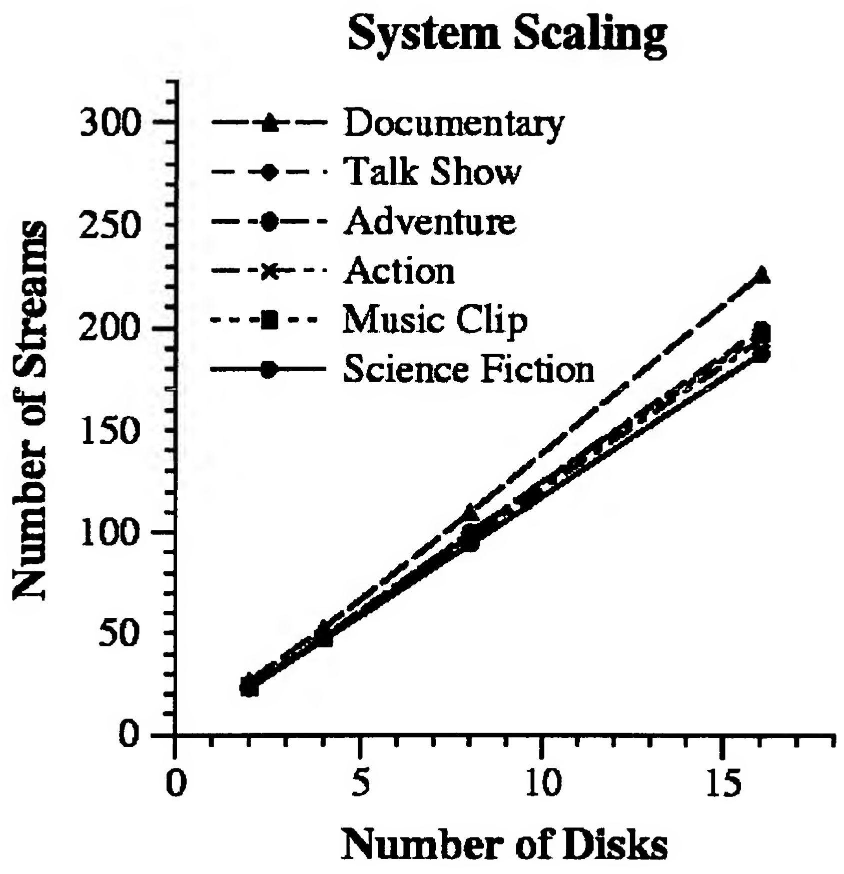

System Scaling
Number of Streams
Number of Disks
Documentary
Talk Show
Adventure
Action
Music Clip
Science Fiction
0
50
100
150
200
250
300
0
5
10
15

Figure 8